Aligning Deos and RTEMS with the FACE Safety Base Operating System Profile

Gedare Bloom
Howard University
Washington, D.C.
gedare.bloom@howard.edu

Joel Sherrill
OAR Corporation
Huntsville, AL
joel.sherrill@oarcorp.com

Gary Gilliland
DDC-I, Inc.
Phoenix, AZ
ggilliland@ddci.com

ABSTRACT

The Open Group Future Airborne Capability Environment (FACE$^{\mathrm{TM}}$) Consortium has developed a reference architecture and standard for real-time embedded avionics systems. The FACE Technical Standard defines required capabilities for real-time operating systems (RTOS), portable components, and a shared data model to facilitate information exchange between components. FACE RTOS requirements are based on ARINC 653 and POSIX 1003.1b with tailoring to address the safety and security needs of avionics systems. Deos is a safety-certified RTOS that supports ARINC 653 but not POSIX. In contrast, RTEMS is an open source RTOS that supports POSIX but not ARINC 653. Integrating a paravirtualized RTEMS with Deos combines the strengths of both and provides a path to conformance with the FACE Safety Base operating system profile. This paper presents the FACE operating system profiles and discusses the technical challenges of the ongoing paravirtualization and integration effort.

CCS Concepts

•Computer systems organization → Real-time operating systems; •Software and its engineering → Real-time systems software;

Keywords

RTEMS, Deos, POSIX, FACE, paravirtualization, avionics

1. INTRODUCTION

The Open Group Future Airborne Capability Environment (FACE$^{\mathrm{TM}}$) is a collaboration of US government and industry to define an open avionics platform suitable for use in both military and commercial aircraft. The Consortium has developed a technical standard and conformance process as well as implementation and business practice guidance. The FACE Technical Standard (Edition 2.1 [8]) defines the infrastructure and application structure required to ensure application portability across platforms. This includes requirements that impact the operating system, programming languages, graphics support, and information interchange between application components.

The FACE Technical Standard defines four operating system profiles, which are a combination of the ARINC 653 [4] and POSIX [3] standards. The profiles are designed to be amenable to different levels of safety and security certification. The capabilities provided by the RTOS are tailored to meet the requirements of avionics systems. ARINC 653 support is required in three of the four operating system profiles. When initially defined, no existing RTOS provided all of the capabilities required to be FACE conformant.

Applications may either be written to the ARINC 653 APIs or to one of the four POSIX profiles. In three of the POSIX profiles, the ARINC 653 services for health, sampling ports, and queueing ports are required to be available for use by POSIX applications while in the fourth profile, this capability is optional. Logically, this is two separate execution environments hosted on an underlying operating system that provides virtualized time and space isolation. There is no requirement that both environments be the same operating system, which led to our approach of using Deos together with RTEMS to meet FACE conformance requirements.

DDC-I and OAR Corporation had been tracking the FACE Consortium effort. But both Deos and RTEMS had significant capabilities missing for conformance. Deos met the FACE ARINC 653 RTOS requirements but did not meet any of the POSIX requirements. RTEMS nearly completely met the POSIX requirements for the single process Safety Base profile but had no ARINC 653 support. After analysis in conjunction with discussions, we decided that executing a paravirtualized RTEMS in a Deos partition was a feasible and affordable path to FACE Safety Base conformance that offered a unique value proposition to end users. Combining Deos and RTEMS capabilities meets most of the FACE operating system profile requirements.

In this paper, we provide an overview of the FACE operating system profile requirements, a discussion of the technical capabilities missing from Deos and RTEMS to meet the Safety Base profile, and the activities required to integrate the two RTOSes capabilities and jointly meet the profile requirements.

2. BACKGROUND

The FACE Reference Architecture is defined in terms of five segments. Figure 1 shows how these segments interface with each other. The Portable Component Segment con-

EWiLi '16, October 6th, 2016, Pittsburgh, USA.

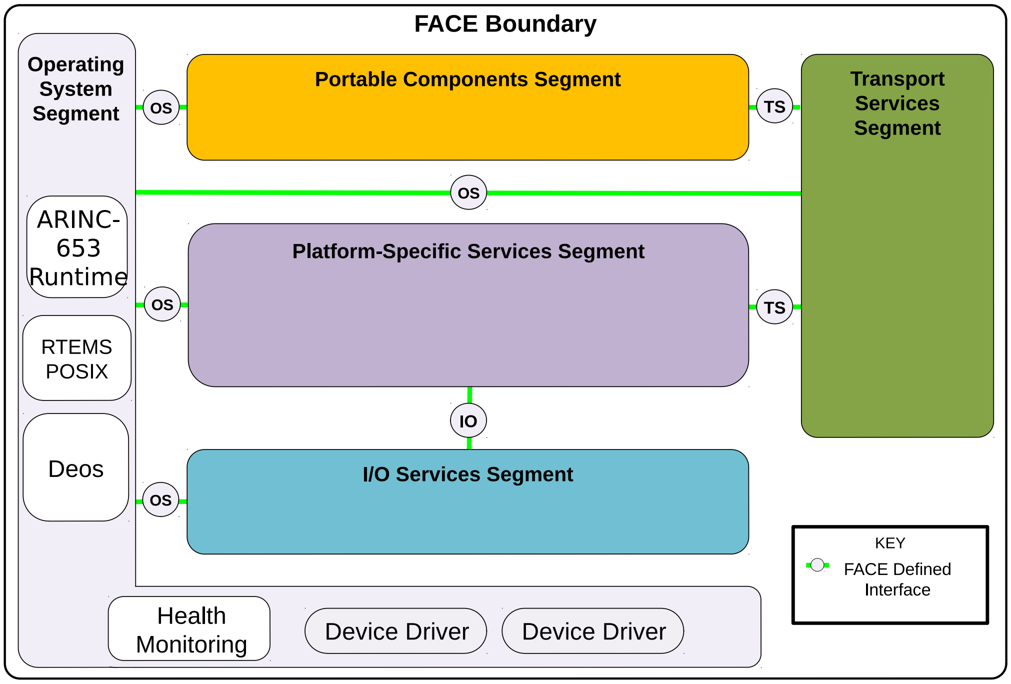

Figure 1: FACE Architecture with Deos and RTEMS in the Operating System Segment (OSS). Other segments shown: Portable Components Segment (PCS), Platform-Specific Services Segment (PSSS) Transport Services Segment (TSS), I/O Services Segment (IOSS).

tains portable business logic as components that are potentially applicable to multiple avionics systems. These components communicate solely using the Transport Services Segment (TSS). The TSS is a critical segment in the FACE Reference Architecture because all application information exchanges are defined in terms of the FACE Shared Data Model and transferred using TSS services. The Platform-Specific Services Segment (PSSS) contains components that include platform specific business logic and device interfaces. PSSS components may interface with both the TSS and I/O Services Segment (IOSS). The IOSS is comprised of software components that provide a bridge between operating system device drivers to the PSSS. The IOSS is accessed only by PSSS components and does not use the TSS.

All of the FACE architectural segments are built on top of the Operating System Segment (OSS). This segment consists of more than the operating system. It includes programming language run-times, frameworks, and health services. However, the foundation for all architectural segments and the deployed platform is the real-time operating system (RTOS) software within the OSS. As shown in Figure 1, the OSS is the only segment that connects with every other segment in the FACE architecture.

2.1 FACE Operating System Profiles

The FACE Technical Standard Edition 2.1 [8] defines four operating system profiles. These profiles were carefully designed to reflect the RTOS services and capabilities that had been through avionics and medical safety and security certifications at various levels of criticality. All of these profiles are subsets of the full POSIX 1003.1b standard, which includes approximately 1300 methods. The FACE operating system profiles are as follows with the smaller profiles being proper subsets of the larger profiles:

- Security - a single process profile that includes 136 POSIX methods and requires ARINC 653. This profile was designed for an application such as an information gateway. Applications written to this profile do not exit; they execute forever. As such, methods to delete, close, or destroy operating system objects are not included.
- Safety Base - a single process profile which includes 246 POSIX methods and requires ARINC 653. This profile also does not include methods used to delete or close operating system objects.
- Safety Extended - a multiprocess, multithreaded profile which includes 335 POSIX methods and requires ARINC 653. This profile also does not include methods used to delete or close operating system objects.
- General Purpose - a multiprocess, multithreaded profile which includes 812 POSIX methods. ARINC 653 support is optional. This profile addresses the needs of applications which do not have safety certification requirements and require richer POSIX support. This profile includes methods used to delete or close operating system objects, and is also the only profile to include `stdin`, `stdout`, and `stderr`.

POSIX partitions hosted on ARINC 653 RTOSes have access to the ARINC 653 health services as well as sampling and queueing ports. POSIX partitions may use shared memory, ARINC 653 ports, or TCP/IP to communicate with applications in other partitions.

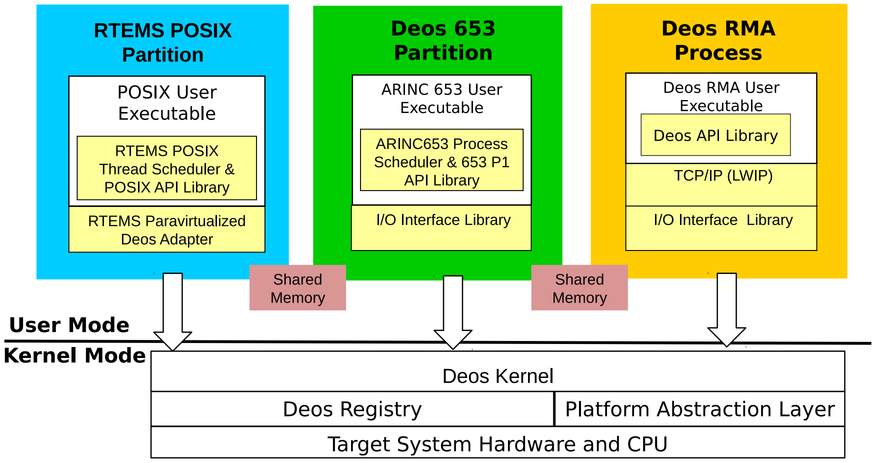

Figure 2: System Architecture Across User-Kernel Modes.

At first glance, the FACE profiles appear to duplicate those defined by POSIX.13[1], but there are significant differences. For example, all FACE POSIX profiles include networking while only two of the four defined by POSIX.13 include it. The largest difference is that the FACE profiles were defined based on industry experience with avionics and other safety qualification. This leads to methods considered unsafe for multi-threaded programming like `strtok()` being included in all profiles defined in POSIX.13-1998 but only in the FACE General Purpose Profile. Even the largest FACE profile, the General Purpose Profile, only includes 812 of the over 1300 APIs in the full POSIX.1 standard. Thus POSIX.13 and FACE are similar yet differ in purpose and content.

2.2 Deos and RTEMS

RTEMS [7] and Deos [5] both have over twenty years of history in the hard real-time safety critical RTOS domain and have evolved to support standardized and non-standard yet commonly used RTOS services. Deos is used in high-end military and avionics safety-critical systems. RTEMS has been deployed in many embedded applications and is best known for its use in the space and physics communities.

DDC-I's Deos is a time and space partitioned DO-178 Level A certifiable commercial off-the-shelf (COTS) RTOS which addresses the high robustness and formal certifiability requirements of avionics and safety critical applications with full support for ARINC Specification 653 Part 1 and Part 4. Deos meets all ARINC 653 requirements for FACE, which is unsurprising since the FACE Technical Standard Security and Safety profiles are based on years of industry experience in developing real-time embedded applications. However, Deos does not have the necessary POSIX support to satisfy the conformace requirements for the Safety Base OS profile.

RTEMS is an open source single process, multithreaded RTOS with robust POSIX 1003.1b support. As a mature, community-developed software system, RTEMS has wide support for about fifteen processor architectures and over 175 board support packages. With user selectable scheduling algorithms, it supports SMP on PowerPC, ARM, SPARC, and x86 and has been tested on systems up to 24 cores. RTEMS includes multiple file systems including in-memory, FAT, JFFS2 and a custom RTEMS File System (RFS) for real-time predictable performance. It also includes FreeBSD TCP/IP and USB support. RTEMS does not have ARINC-653 support so it cannot alone satisfy the Safety Base OS profile, and yet when initially evaluated, RTEMS was missing fewer than 10 of the 246 POSIX methods required by this profile.

Interestingly, evaluated against the multiprocess FACE General Purpose OS profile RTEMS does surprisingly well, supporting approximately 90% of the methods required. Required capabilities such as `fork()/exec()` and process groups are beyond the single-process target profile for RTEMS. However, many of the methods required by the FACE profiles are not thread concurrency and synchronization related, but instead are part of the Standard C Library. RTEMS uses the Newlib C Library, which is also used by Cygwin. Most of the missing methods do not require a multiprocessing operating system. Newlib does not support `<fenv.h>` or long double complex math, and these account for most of the missing methods that RTEMS could support.

3. ACHIEVING FACE CONFORMANCE

In order to meet the requirements of the Safety Base OS profile, we chose to paravirtualize RTEMS for efficient execution within a Deos partition, or container, that provides the required POSIX services. Applications therefore will be partitioned by Deos with ARINC-653 tasks, native tasks scheduled according to the rate monotonic algorithm (RMA), and POSIX application threads or services that are accessed through calls into the RTEMS partition. Figure 2 depicts this partitioning and the placement of the user-kernel boundary in the combined system. Some of the required POSIX services were missing from RTEMS, and we have started to add those services as well. Beyond conformance this effort is also aiming to promote a positive user experience by integrating the development environments of the two RTOSes.

3.1 Paravirtualizing RTEMS for Deos

Figure 3 shows how Deos schedules the POSIX applica-

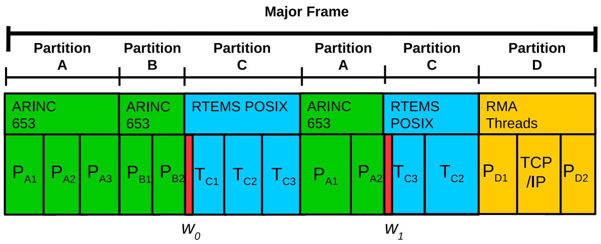

Figure 3: Scheduling of threads across ARINC-653 partitions (provided by Deos), RTEMS POSIX environment, and other rate monotonic (RMA) tasks inside Deos. At the start of each RTEMS partition a time budget exceeded exception is delivered from Deos to RTEMS and used by RTEMS to update time accounting. The elapsed time between RTEMS partitions is calculated by subtracting the RTEMS time budget from the inter-partition scheduling gap $(w_1 - w_0)$.

tions in the RTEMS partition concurrently to the ARINC-653 and other native tasks. The beauty of this design is that core attributes of both RTOSes are preserved. Deos is unmodified and retains its Level A certifiability. RTEMS remains an open source RTOS with all of its capabilities intact. Moreover, each RTOS can be used independently just as they can now. In fact, within limits, bare metal RTEMS can host FACE conformant applications that can be easily migrated to the fully conformant Deos/RTEMS environment.

RTEMS paravirtualization support had to be provided for Deos as well as a Deos run-time adapter layer. The paravirtualization support allows RTEMS to execute in user space (i.e. no privileged or sensitive instructions) and provides critical sections, proper memory layout, and interactions with the Deos kernel for timing and I/O. Since Deos and RTEMS both use the GNU tools for cross development and the Executable and Link Format (ELF) object format, it was straightforward to compile an RTEMS executable which could be loaded by the Deos loader. Initially implemented on the x86 architecture, the Deos adapter layer currently maps video memory for framebuffer output and a COM port for serial I/O to the RTEMS partition.

One of the design choices was whether or not the RTEMS IDLE thread should yield the time window back to Deos. In an SMP system, a single core going idle does not imply that all cores are idle, and detecting when all cores go idle may be impractical to detect on an SMP Deos partition. Even on a uniprocess system, it is possible that a periodic task should release a job during the current partition execution window. Thus, RTEMS partitions consume the entire allotted execution time. This results in the partition always ending with a time budget exceeded exception. As we discuss next, this exception also simplifies the time management across the RTEMS and Deos layers.

A critical issue to the correctness of the integration of the two RTOSes is the handling of time. RTEMS normally executes in bare metal environments, is fully aware of the passage of time, and goes to great pains to manage time correctly. In a time and space partitioned environment, an RTEMS partition will be unaware of how much time has passed while it was not executing. Partitions in an ARINC 653 environment execute in an order specified by the system integrator with no concept of the passage of time while they are not running. This lack of global time knowledge in RTEMS leads to the following problems:

1. Time keeping mechanisms rely on periodic clock tick interrupts.

2. Accurate timestamps require knowledge of length of time since last clock tick.

3. Obtaining precise interval timeouts (e.g. delay for fifty milliseconds)

4. Maintaining the time of day.

5. Keeping CPU usage statistics for each thread that executes.

6. Tracking wall time for rate monotonic periods each periodic task.

To deal with these problems, RTEMS needs a few simple mechanisms to account for the accurate passage of time. The approach we are taking is to consume all of the budgeted time given to the RTEMS partition, which will cause a "time budget exceeded exception" to be delivered at the start of each new scheduling window for the partition. This exception is where the passage of time is tracked. During RTEMS initialization, the Deos adapter clock tick device driver obtains an initial uptime. The exception handler obtains the uptime from Deos and calculates the length of time that this RTEMS partition was not executing. As a side effect of this design choice, RTEMS threads that block for any service will not unblock during the same scheduling window. The RTEMS clock driver obtains the Deos uptime at any point to track passage of time during RTEMS execution.

One challenge introduced by the time management approach is that variable lengths of time elapse when RTEMS is not executing, and that elapsed time could cause more than one clock tick to expire. Hence, we are carefully studying the effects of variable length ticks and multiple ticks on the kernel services of RTEMS.

Another challenge is that task executions may span scheduler windows. For example, in Figure 3 the execution of T_{C2} in the figure is entirely within a single time window and RTEMS can know both the starting and ending execution time for that thread. However, the execution time for T_{C3} is at the end of one window and the beginning of the next with at least one other partition execution between those two fragments. The exception-based variable clock tick based on the beginning of time windows does not mark the end of a time window. To overcome this challenge, RTEMS gets the uptime in nanoseconds at the start of a window (w_1) and deducts the elapsed time at the start of the previous window (w_0) to determine the inter-partition scheduling gap $(w_1 - w_0)$ inclusive of the execution time of the previous window. To also remove that execution time simply deduct the budget.

The FACE Technical Standard includes two requirements related to setting the date and time. First, the RTOS must support configuring which partitions are allowed to set the date and time. Second, setting the time must set the date and time on all partitions in the system. The first requirement results in Deos having a configuration parameter to determine if a partition can set the time. This configuration information is used to reject attempts to set the time when a partition does not have sufficient permission. The second requirement imposes changes on RTEMS, and possibly Deos. In the normal bare metal configuration, RTEMS does not have to pass the date and time set operation to another RTOS. In the paravirtualized Deos/RTEMS environment, RTEMS has to reflect the set operation to the Deos kernel. If Deos allows this partition to set the time, then the other partitions will be notified of the time change when they execute. For RTEMS to know that the underlying Deos time has changed we consider adding a generation counter to the Deos time and ensuring that the RTEMS partition checks this at the beginning of each execution window. This requires an addition to Deos. Alternatively, since RTEMS knows the elapsed time between windows by the calculation of $(w_1 - w_0)$, then RTEMS could also store the previous underlying Deos wall time and compare that with the current Deos wall time plus the inter-partition scheduling gap to detect that the Deos wall time has been changed while the RTEMS partition was not executing.

3.2 Missing POSIX Services in RTEMS

The required POSIX services missing from RTEMS could be broken into two categories. The first category includes services that require a memory management unit (MMU) and virtual memory environment. RTEMS does not have virtual memory and only has limited support for MMUs. When hosted in a Deos partition, RTEMS must integrate with the underlying kernel to provide POSIX shared memory and memory mapping. The second category of missing services are those that could be fully supported in the normal, non-MMU RTEMS execution environment. These services just have not been demanded yet by the RTEMS user community.

The FACE Safety Base operating system profile requires the `shm_open()` and `mmap()` methods but does not require the `shm_unlink()`, `munmap()`, `mlockall()`, `munlockall()`, `mlock()`, `munlock()`, or `mprotect()` methods. Deos natively supports named shared memory and mappable memory regions, which are statically configured by the system integrator. RTEMS is being augmented to provide the POSIX services by calling through the paravirtualization layer to access the Deos kernel specific memory management services. Although not required, `shm_unlink()` and `munmap()` are natural to provide. The memory locking and protection APIs may be provided in the future, but since global memory objects and their accessibility are statically configured and there is no paging of memory to storage, they provide little value.

The first missing POSIX APIs added to RTEMS were `pthread_getconcurrency()` and `pthread_setconcurrency()`. These are rarely used methods that specify the desired mapping of user threads onto kernel threads. RTEMS implements a 1:1 mapping. The POSIX standard is very specific on the implementation when an operating system directly maps user threads onto kernel threads. The value set by `pthread_setconcurrency()` is to be returned by `pthread_getconcurrency()` and otherwise have no impact. This was trivial to implement and is fully supported in the normal bare metal RTEMS usage.

The next missing service was `pthread_setschedprio()`. This method is used to set the priority of a thread without the side-effect of yielding the processor to threads of the same priority. It also provides a simplified interface compared to `pthread_setschedparam` which supports changing the scheduling policy and parameters of a thread. RTEMS already had internal support for altering a thread's priority without yielding and `pthread_setschedprio()` was similar to the Classic API method `rtems_task_set_priority()`. This methods was trivial to implement and is fully supported in the normal bare metal RTEMS usage.

The missing methods to associate a POSIX clock with a condition variable proved more complicated to implement. The `pthread_condattr_getclock()` and `pthread_condattr_setclock()` methods are themselves trivial to implement, but the implication is that timeouts on a condition variable can be based on either wall time (e.g. `CLOCK_REALTIME`) or absolute time (e.g. `CLOCK_MONOTONIC`). Most RTEMS time outs are in terms of intervals that are logically comparable to the intent of using `CLOCK_MONOTONIC`. However, the condition variable manager in RTEMS did not support multiple clocks and support had to be added. Also, the POSIX timer and clock services had to be reviewed in regards to their support for multiple clocks and fixed where necessary. Additionally, until recently, internally all time events using wall time had a granularity of one second. Coincidentally, this restriction was only recently removed as part of improving time management for SMP systems.

The FACE operating system profiles include the `posix_devctl()` method, which is from POSIX 1003.26 [2]. This method provides a standard alternative to `ioctl()`, but neither GNU/Linux nor FreeBSD implement it. At this time, no work has been performed implementing this method, which must be provided for FACE Safety Base conformance.

3.3 Development Tools

No matter how technically capable the run-time environment is, the user's experience is driven by how well the RTEMS POSIX support is integrated with the native Deos development environment. RTEMS applications will be compiled using the normal RTEMS GNU toolset but using the OpenArbor Eclipse IDE. This IDE now provides templates for both native Deos and RTEMS POSIX applications. It supports configuring partition time and space attributes as well as the partition execution schedule. OpenArbor supports timeline visualization and inspection of Deos run-time characteristics. That is, it allows the user to view application status at a given point in time. That status includes several types of information for RTEMS and Deos:

- Processes: Users can view the set of processes that are active and various attributes of each process, including the threads it contains and quotas in terms of budgeted resources versus actual usage (e.g., RAM, semaphores, mutexes, events, etc.).

- Threads: Users can view the set of threads that are active on a global basis or on a process by process basis including various attributes of each thread such as execution rate, execution time budgeted versus time actually used in the period being viewed, stack size budgeted versus stack actually used, exceptions experienced by the thread, and events logged by the thread.

3.4 Performance

At this time, no performance measurements have been gathered. However, no modifications have been made to Deos as part of this integration effort. Therefore, its performance will not change.

The paravirtualization of RTEMS does result in minor changes versus a bare metal configuration. The primary change from a performance viewpoint is that the interrupt disable/enable methods cannot use supervisor instructions and thus must be adapted to user space in the partition. However, the implementation is very light using atomic synchronization instructions instead of disabling interrupts. Performance should be comparable to the bare metal alternative of disabling maskable interrupts via special instructions. Clock ticks occur as a side-effect of the Deos time window exceeded exception and these are processed at the beginning of each execution window. If this exception occurs during the middle of an interrupt disable critical section, the clock tick processing is deferred and performed as part of enabling interrupts.

Otherwise, the paravirtualization is focused on I/O and memory layout in the partition, which has no impact on performance of concurrency and synchronization primitives for managing threads.

4. RELATED WORK

Closely related is the AIR and AIR-II software architectures that provide an ARINC-653 environment by using the AIR Partition Management microkernel as the privileged kernel and adding an application executive (APEX) Layer to translate APEX calls into POSIX or native RTOS system calls so that APEX services can be satisfied by COTS RTOS software to be ARINC-653 compliant. We focus on FACE compliance, and are also pursuing the use of interpartition communication to access required services satisfied by the RTEMS kernel on behalf of applications executing in a (non-RTEMS) partition. Also related to this project is the use of virtualization to implement a separation kernel, an approach used to support integrated modular avionics by PikeOS and XtratuM [10]. These systems have not demonstrated FACE compliance, and they do not put required services in a guest partition. Other virtualization approaches that use a designated virtual machine to export sensitive services and manage the hypervisor, e.g. Xen's dom0, are related but not applicable to the avionics application domain. Separation kernels are also prevalent in alternative approaches to provide complementary safety and security mechanisms, the most prominent being MILS [9]. Since MILS provides strong security guarantees especially related to confidentiality and integrity, it is not plausible to export services between partitions without substantial effort to certify or monitor the information flow between them.

5. CONCLUSION AND FUTURE WORK

At this point, RTEMS applications can be executed in a Deos partition and most of the FACE operating system requirements are met. There is still implementation work underway with current RTEMS-related activities focused on RTEMS support for POSIX shared memory and mmap APIs, ensuring that timing in the paravirtualized environment reflects external reality, and porting beyond x86 initially to the PowerPC, which is commonly used for avionics. All improvements to the RTEMS POSIX implementation have been merged with the RTEMS public git repository. No specific modifications to Deos have been made at this time, although we anticipate modifying the Deos Lightweight IP stack (lwIP) TCP/IP stack[6] to meet FACE operating system networking requirements. Most Deos activities have been focused on integrating RTEMS POSIX application support into the OpenArbor Eclipse IDE and identifying opportunities for tighter integration between the RTEMS POSIX environment and the IDE that could improve the user experience. There is significant industry interest in the combined Deos/RTEMS environment as it becomes more aligned with the FACE Technical Standard. When Deos/RTEMS is completely aligned with the FACE Safety Base Operating System profile, it will be submitted to a FACE Verification Authority for conformance verification review.

6. ACKNOWLEDGMENTS

FACE is a registered trademark of The Open Group. Deos and OpenArbor are registered trademarks of DDC-I, Inc. RTEMS is a trademark of Online Applications Research Corporation. This project is supported by the author's respective employers, however any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of their employer(s).

7. REFERENCES

[1] Ieee standard for information technology - standardized application environment profile - posix(r) realtime application support (aep). *IEEE Std 1003.13-1998*, pages i–, 1999.

[2] IEEE standard for information technology - portable operating system interface (POSIX) - part 26: Device control application program interface (API) [c language]. *IEEE Std 1003.26-2003*, pages 0_1–31, 2004.

[3] Standard for information technology portable operating system interface (POSIX(R)) base specifications, issue 7. *IEEE Std 1003.1, 2013 Edition (incorporates IEEE Std 1003.1-2008, and IEEE Std 1003.1-2008/Cor 1-2013)*, pages 1–3906, April 2013.

[4] ARINC Specification 653 P1-4, Avionics Application Software Standard Interface, Part 1: Required Services, 2015.

[5] Deos: A time and space partitioned DO-178 Level A certifiable RTOS, 2016. http://www.ddci.com/products_deos/.

[6] The lightweightip stack (lwIP), 2016. http://www.nongnu.org/lwip.

[7] RTEMS real time operating system (RTOS), 2016. https://www.rtems.org/.

[8] Technical standard for Future Airborne Capability Environment (FACE), edition 2.1, May 2014. http://www.opengroup.org/face/tech-standard-2.1.

[9] J. Alves-Foss, P. Oman, R. Bradetich, X. He, and J. Song. Implications of Multi-Core Architectures on the Development of Multiple Independent Levels of Security (MILS) Compliant Systems. Technical report, Oct. 2012.

[10] A. Crespo, I. Ripoll, M. Masmano, P. Arberet, and J. Metge. Xtratum an open source hypervisor for tsp embedded systems in aerospace. *Data Systems In Aerospace DASIA, Istanbul, Turkey*, 2009.